ASTEROID MINERS

A UNIQUE GAME TUTORIAL

By Mark Chasin

CONTENTS:

ASTEROID MINERS

A UNIQUE GAME TUTORIAL

by Mark Chasin

1. Introduction

You have just purchased a remarkable teaching aid for ATARI BASIC, one you can learn from, but also PLAY as well! Many of the more complicated and least well-documented features of the ATARI are exploited in this program, and reading this book will give you the insight you will need to make these techniques work for you, as well. These include multiprocessing using the vertical blank interrupt interval, machine language movement of players in player-missle graphics, the use of more than 25 representations of players, multiple players used together to form a multicolored and multifunctional player, and a redefined character set, including the machine language routine to move the set from ROM to RAM to allow you to change it! Once you understand these techniques, you should be proficient at producing really fantastic effects. Remember, MMG MICRO SOFTWARE is very interested in reviewing any of your efforts you feel are marketable. Write to us for details.

This program has been compacted using the Masher program available from the ATARI Program Exchange. This combines lines and converts the most frequently used constants to variables. Constants are stored in the ATARI as six byte, floating point numbers, and take more storage and execute far more slowly than variables. These variables, beginning with the letter Q, can be found on lines 110 and 120 of the program. If you need to translate the variables back to numbers, look on these lines for the coding scheme.

As the discussion of the program proceeds, you will need to refer to the BASIC and assembly language listings of the source code for each part of the program. These can be found following this discussion, and are thoroughly commented for ease of understanding.

2. Organization

ASTEROID MINERS is arranged with the main loop which controls the play of the game itself located from lines 1360-1680,roughly in the center of the program. The frequently called subroutines are placed at the beginning of the program, and the infrequently called subroutines are found at the end of the program. This is the most efficient way of calling subroutines, since ATARI BASIC searches for a subroutine when a GOSUB command is encountered by starting at line 0 and continuing through the lines until the correct line number is found. If frequently called subroutines are placed at the end of the program, they run much, much more slowly. If you want to experiment, put the subroutine to read the joystick, now at lines 350-420, at the end of the program, and see how much more slowly the program executes. Since this program was designed as a teaching aid, there are many lines of comments throughout the program. If you would like the game to run faster, remove these comments. This is a general rule for BASIC, and can be used for your own programs, as well.

Throughout the program, you will find a line that reads:

```
REM POKE 16,64:POKE 53774,64
```

This line disables the BREAK key, so that the program cannot be interrupted while it is in progress. To be effective, you must remove the REM, converting this from a REMARK to an active BASIC statement. However, each time the screen is cleared, the BREAK key must be disabled again. That is why there are so many of these statements throughout the program. Finally, in the fourth line of the program, the command POKE 580,1 can be found. This command changes the SYSTEM RESET from a warm start to a cold start condition. In English, this means that instead of simply resetting the computer and returning you to the READY prompt, the SYSTEM RESET button now acts as if you had just turned the computer on, and your disk drive will reboot, if on. You will LOSE whatever is stored in the computer at that point, so perhaps you will wish to remove that statement from the program. The combination of disabling the BREAK key and changing the SYSTEM RESET to a cold start condition is an effective way of protecting your code from the prying eyes of others, if you so desire.

3. The Main Game Loop

Before we start a discussion of the main loop, it will be necessary to discuss the use of player-missle graphics in this game. First, the mining ship is actually 3 players in one, the blue ship, the red flame, and a black figure in between the arms of the ship, which is used to detect collisions between the asteroids and this area between the ship's arms, indicating a correct pickup by the ship. Collisions between the ship itself and the asteroids result in explosions, so it is important to detect a "clean catch" as contrasted with a true ship-to-asteroid collision. The result of this is to require three moves for each change of ship position, both in the X and Y direction,since the ship is really three players.

The main loop begins by storing the old Y value in BB, the old X value in CC, and the old joystick position, J, in JJ. Then we determine if time has run out; if it has, PEEK(1619)=1, and the programs jumps to the routine at line 2250. POKE 53278,1 clears the collision registers in player-missle graphics, and the other three POKEs in line 1370 set the X position of the ship. Line 1380 gets the position of the joystick, and, if the button is pressed, makes each move 4 spaces instead of 1, speeding up the ship's travel. If the position of the joystick hasn't changed from the last loop, we GOSUB 350 because we only have one dimension of travel to worry about, and the ship is still facing in the same direction. In line 1400, if the joystick is straight up, the ship is stopped, so we turn off the sound of the engines, and erase the flame. Then we start the main loop again, since we can't hit anything standing still.

POKE 77,0 resets the counter which would cause the screen to rotate colors. Then we get the movement parameters from 510. Lines 1420-1440 erase the old ship, and draw the new position, facing in the right direction. Line 1460 tests the collision registers. If P1 (ship-to-asteroid) has occurred, and if the difficulty is 2 or 3, then an explosion occurs (to line 2640). If no collision at all has occurred (line 1480), then the program begins the main loop again. Otherwise, we need to calculate the position at which the collision between the detector and the asteroid has occurred using GRAPHICS 0 (playfield) coordinates. Line 1490 converts these for us, and line 1500 determines which color asteroid we've gotten, and its value. If it's not an asteroid (32) then we'll begin the main loop again; otherwise, we'll erase the asteroid (line 1530) and make the appropriate sound, depending on its value. In line 1560, we decrease the number of good (blue or gold) asteroids by one, and then we add the approriate number to the score, and print it to the screen. If there are still some good asteroids left to collect, we'll go back to the start of the main loop again.

Otherwise, we'll stop the clock by using the machine language subroutine at memory location 1762 (line 1580), sound the trumpets to signal the clearing of a board, move the ship to X=0 (remove it from the screen with 3 fast pokes), refill the screen with asteroids (subroutine at line 1820), reset the X and Y positions of the ship to the starting positions, redraw the ship at these coordinates, add a bonus at line 1630, restart the clock at line 1670, and then start the main loop again. Line 1660 does the actual draw of the ship, facing down, and sets the Direct Memory Access Control (location 559) to allow the ANTIC chip to get its information directly from memory, enables player-missle graphics, and sets the playfield width to normal. Location 623 sets the priority of the ship and the asteroids, to determine which will appear in front of the other when they meet.

4. The Subroutines

a. Line 350

The first four lines determine whether X or Y has changed, and change the position accordingly. Note that the position is changed by adding or subtracting MOV, which can be either 1 or 4, depending on whether or not the button is pressed. This gives two speeds to the ship. The last four lines check to make sure the ship is still within the confines of the asteroid field on the screen. They actually prevent the ship from leaving the field of play.

b. Line 510

This is similar to that described above, but since the direction of travel has changed, we must change the way the ship is facing, as well as its X and Y coordinates. The first 8 lines handle this, and the last four function as above, to restrict the ship to the field of play.

c. Line 710

After asking for the degree of difficulty at which the game is to be played, the program OPENs the keyboard for input, and GETs the answer. Note that the use of GET, rather than INPUT, requires an OPENing of the keyboard, but does not require that the users follow their choice by a

RETURN. In general, it is easier for the user if you use GETs, rather than INPUTs, although this only works for single character answers. By GETting each character of a multicharacter input, and storing them in a string, you can even GET a whole string, without requiring a RETURN, if you know how long the string is to be. If the answer is not valid, the program asks again. In lines 750 and 760, an array of numbers is established, for the white, blue and gold asteroids, at each of the three levels of difficulty. Lines 770 through 790 establish how many white (SPLIM), blue (ALIM) and gold (BLIM) asteroids belong in the field, and the time and difficulty are set, as well.

d. Line 900

This routine simply sets up the frontpiece of the game in GRAPHICS 2. It takes advantage of the split screen mode, setting the background of both the graphics and text windows to black. The screen appears to be all one mode, but the letters at the top of the screen are large, colored, GRAPHICS 2 letters, and the author's name and the copyright notice are in small, white, GRAPHICS 0 letters at the bottom of the screen. This could have been switched around, but would have required a modified display list. The technique used here, for this purpose, is just as effective, and far easier.

e. Line 1000

We begin by relocating the character set at machine language speeds, setting up the frontpiece, and playing 2001 - A Space Oddysey, to get the player in the proper mood. Line 1010 asks whether the player needs instructions, checks for a proper reply, and responds accordingly. Note that we use a GET here, as we did above, but we specify that the answer should be either Y/N.

Next we overwrite the star field with some introduction to the game. An important point to note here is that this introduction actually hides the access to several of the other subroutines required before play can begin. These can be seen on lines 1050 (GOSUB 3680), 1110 (GOSUB 3900) and 1140 (GOSUB 3970). These routines POKE the player's shapes into memory (3680) and the countdown timer and movement subroutines (3900 and 3970) as well. Normally, when you print to the screen, you need a delay loop to leave the message on the screen long enough for the user to read it. The routines accomplish this, but also do some productive work as well. In general, you should try to make anything that will take time do something productive for you, or it simply becomes a lag in your program.

In line 1160, we go get the level of difficulty to be played, ring the bell once, and in the wait for the next bell tone, poke in the new character shapes into our relocated character set. In the next line, we set up the top 8 pages of remaining memory for the player-missle storage area. Note! remaining memory. The top 4 pages of the memory we started with are now occupied by the redefined character set. POKE 54279,A tells the computer where we have put the player-missles. POKE 53277,2 enables the players. POKEing 1 here would enable the missles, which are not used in this game, and POKEing 3 here would enable both players and missles. Since the units of variable A are in pages, we multiply by 256 to get the actual address of the beginning of the player-missle storage area. POKE 53256,1 sets the size of the ship, and the last three variables in this line are the starting X and Y coordinates and the initial direction of travel of the ship.

In line 1180, the colors of the ship, flame and indicator player are set. To set these, we POKE the color registers for players 0, 1 and 2 with sixteen times the color, plus the luminance. For example, the blue of the ship is color 7, luminance 4, so we POKE 16*7+4=116. Likewise, the flame is red, so we POKE 16*4+4=68. Since we don't want the indicator player to be seen, we set it to pitch black, against the black background. Then we ring the bell a second time, to let the player know we're still around, and POKE the pointers to the locations in memory where each of the shapes of the ship and other players are found (GOSUB 2140). Remember that the ship can go in any of eight direction, so we need the shapes of 24 different players stored in memory. Three more players are stored by the subroutine at line 2640, for a grand total of 27 players!

In line 1190, the machine language call clears out the whole player-missle storage area to all zeros. If we didn't do this, strange shapes would appear when we activated the player-missle graphics. Normally, in BASIC, this clearing would take from 20-30 seconds, but at machine language speeds, it's over in the blinking of an eye. From here to line 1230, we set up the playing field, and insert the asteroids (GOSUB 1820 in line 1240). We then ring the bell again, set the size of the flame and indicator player, print the high score and put the ship into the player-missle storage area. POKE 559,42 and POKE 623,1 are described above under the Main Game Loop section. In line 1280, the three POKEs set the initial time on the countdown timer, and the machine language call USR(1622) turns the timer on, just before the main loop starts.

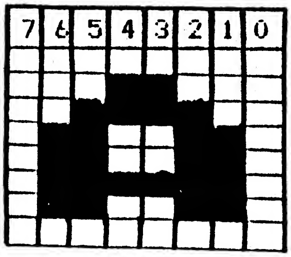
f. Line 1820

This subroutine is divided into 3 loops, each of which place a different colored asteroid into the field. Each loop checks before placing an asteroid into the field to be sure it isn't erasing a valuable one, and isn't placing one in the area to be occupied by the ship in its initial position. Line 1940 clears out this area, so that the ship doesn't explode as soon as the game starts. Note that NUMGOOD is the total number of blue and gold asteroids, so that the computer can keep track of the progress being made in clearing the screen of these.

g. Line 2030

This subroutine redefines four characters, the minus sign and the three shapes of the asteroids. The colors of the asteroids come from artifactual colors. That is, to make a white line in GRAPHICS 0, your ATARI draws a blue line next to a gold line, and the combination appears white. Therefore, the white asteroid has all of its lines drawn in. However, the blue and gold asteroids only have lines drawn on the odd or even bits.

Each character in the ATARI character set is drawn in a box measuring 8 bits wide, by 8 bytes high. In fact, only the center 6 bytes are used in the vertical direction and the center 6 bits in the horizontal direction, so that the letters don't touch each other when they are printed adjacent to one another. These 36 bits then determine the shape of the character to be printed. Each bit turned on results in a dot on your TV screen. For example, an A looks like:



Therefore, to produce blue and gold objects, you only need to design your shape using either bits 7,5,3,and 1, or 6,4,2, and

0. To convert these bit patterns into numbers to POKE into memory, as is done in this subroutine, simply refer to the chart below, and add up all of the bit values for each byte.

BIT #:	7	6	5	4	3	2	1	0
VALUE:	128	64	32	16	8	4	2	1

So, for the third byte of our letter A, the value would be 32+16+8+4, or 60. Just calculate this value for each byte of your character, and POKE it into the the spot of your choice in the relocated character set. Remember, the character set used is the internal set, found on page 55 of the ATARI REFERENCE MANUAL.

h. Line 2140

As described above, this program uses many players. However, the ATARI only has the capability of dealing with four (or five, under special circumstances) players at a time. So how can we deal with so many? The trick is to use only three at a time, which is well within the power of the computer. As you know from the discussion above, the shapes of the players must be stored somewhere in memory. A thorough discussion of player-missle designing is beyond the scope of this tutorial, but it will be reviewed briefly.

Essentially, the process is identical to redefining a character. A bit map of the shape of the player is first drawn. One difference, however, is that whereas a redefined character must be 8x8, a player can be 8 bits wide by a very large number of bytes high. How high? It is possible to create a single player that would occupy the entire screen from top to bottom. Does that give you any ideas? In any case, once the bit map is complete and put into memory, it is most frequently dealt with by directly refering to it when it is needed. ASTEROID MINERS uses a technique first suggested by Eric Stoltman in COMPUTE!, October, 1981, Issue #17. This technique sets up pointers to multiple representations of a player in memory, and simply accesses the pointers.

As an example, the rocket ship used in ASTEROID MINERS, as was discussed above, is actually three players drawn together. The ship can move in eight directions, up, down, left, right, and the four diagonal directions. For each direction, the ship actually has a different shape, i.e., when going left, the flame is on the right and the scoop is on the left, etc. Line 2140 sets up pointers that tell the program where to find each shape of the ship itself. Line 2150 sets up the pointers to the eight possible flames, and line 2160 to the eight possible indicator players. You will note that there are really nine pointers for each player. The ninth points to a blank area of memory, and is used to point when the program needs to erase the player. Each pointer indicates a specific shape, from zero, which points to the player moving straight up the screen, through 2 (right), 6 (diagonally down and to the left), to 8. So, if the ship must be drawn moving straight to the left, each of the pointers is set to 7, and the correct shape will appear on the screen. This assignment is made using the variable FACING throughout the program.

i. Line 2250

The subroutine at line 2250 sets up a new screen each time the game ends. The rocket engine sound is turned off and the score is checked to see if a new high score has been established. The clock is stopped in line 2260, and the top portion of the screen is reprinted on the screen. Line 2300 moves all three players (the ship) to the horizontal position of zero, off the screen. This is an effective way of erasing them very quickly, although they will need to have their horizontal positions reset using this technique. Setting FACING to nine and redrawing the players is only a little slower, but doesn't require a repositioning, since the player is still positioned, but "undrawn". Line 2310 removes any remaining asteroids from the screen, OPENs the keyboard for the GET that's coming, and resets the color registers so that what is on the screen becomes visible. If no further game is desired, the screen is cleared with the GRAPHICS 0 command,and the program is ERASED from memory by the NEW command. If you would like it to remain in memory after ending, either remove the NEW command in line 2320, or hit the BREAK key at any time during the program. If another game is desired, the GOSUB 720 determines the degree of difficulty of the new game. This is important, since it allows a player to practice at one level of difficulty, and then switch to another without restarting the game. This feature should be included in all programs, wherever possible. The remainder of line 2330 sets up the conditions for restarting the game, and erases the old score. Control is then returned to the start of the main loop.

j. Line 2420

To maintain interest in a game, several levels of difficulty are generally a good idea, since players can then work their way up in difficulty. ASTEROID MINERS has three such levels, and in the highest level, a random factor is added. Periodically (each time the countdown timer registers that one more minute has passed), a new blue or gold asteroid is added to the screen in a randomly selected position. This serves to make the game more difficult in two ways. First, if it appears in the same location as the current position of the ship, BANG! Secondly, the player may have just about cleared the screen, when suddenly another asteroid appears in some difficult-to-reach position. Obviously, this concept could be extended to making a very difficult game, by adding asteroids more frequently, but mercy is called for here. The routine at line 2420 simply adds the new asteroid, and increases NUMGOOD. Note that line 2420 checks to see that the randomly selected location for the new asteroid doesn't already have one. If it does, a new location is selected before the asteroid is PRINTed to the screen.

k. Line 2510

In general, a reward for accomplishment should be included in all endeavors. ASTEROID MINERS rewards a successful clearing of a screen in two ways, a bonus score proportional to the level of difficulty, and a heralding sounding of trumpets. This subroutine turns off the sound of the rocket motors, plays the trumpets, and then turns off the trumpets.

l. Line 2640

In keeping with a reward for successful performance, failure should also be dealt with accordingly. In ASTEROID MINERS, at any level above the training mode (level 1), if an asteroid is not cleanly scooped up, the ship explodes by this subroutine.

Line 2640 moves the ship off the screen, and brings on a new player, the explosion! Its color-luminance is set with the POKE to 707, as described above, and the shape of the mushroom cloud is POKEd into memory. The POKE to 623 is the priority select, and the POKE to 53251 positions the mushroom cloud where the ship used to be. The loop from 2670 to 2710 puts the sound of the explosion out, flashes the screen in an ever changing spectrum of colors, and changes the size of the cloud twice (more players!) during the explosion, giving the impression of a spreading mushroom cloud. After the loop, the colors are reset, the sound is turned off, the cloud is moved offscreen, and we find out if another game is desired by the GOTO 2250.

m. Line 2800

In order to redefine a character set, we must put new bit patterns into the old set, as described above. However, the character set is located in ROM, Read Only Memory. We cannot write to this, so we can't change it. In order to accomplish this change, we must first move the whole character set from ROM to RAM. This can be done with a simple loop in BASIC, and needs to move 128 characters, with 8 bytes per character, for a total of 1024 bytes. However, this loop in BASIC takes over 20 seconds to execute. The same transfer can be done in machine language in far less than a second! The machine language program to do this is stored in the string QWER$ on line 2810, and is activated by the USR call at the end of the line.

Before we can move the character set to RAM, we need a protected place to store it. Line 2800 moves the top of memory pointer (location 106) down four pages in memory. Note that each page of memory holds 256 bytes, and 4*256=1024, which is exactly the space we need for the whole character set. We then need to tell the computer where to find the new character set, and this is done by the POKE 756. THIS POKE MUST BE REPEATED EACH TIME A NEW GRAPHICS MODE IS CALLED! This is because the ATARI resets this pointer to its original place pointing to the original ROM position after each GRAPHICS call.

For discussion of this machine language routine, please refer to the appropriate assembly language program enclosed, titled Character Set Relocating Routine. The origin is set to $0600 here, but since it is fully relocatable, we can store it as a string and let BASIC worry about where to put it. By the way, to create this routine, use the Editor/Assembler cartridge, and, after you assemble it, SAVE it to disk. Then put in BASIC, and, from DOS, use the L option (binary load) to put the routine in memory. Then hit B to get back to BASIC. You now have your machine lanquage program stored in 34 memory locations, starting at 1536 (1536 decimal = $0600). In the direct mode, type:

```
DIM A$(34):FOR I=1 TO 34:X=PEEK(1535+I):
        A$(I,I)=CHR$(X):NEXT I
```

This will create a string with your machine lanquage code in it. Then, still in the direct mode, type:

```
PRINT A$
```

After the string is on the screen, move the cursor to add a line number in front of it, and after the line number, type:

```
DIM A$(34):A$="
```

and after the string, put another quote. When you hit RETURN, your machine language subroutine is part of your program!

The subroutine to relocate the character set from ROM to RAM begins with PLA, which is required for access by BASIC. The next four instructions take the location to which the character set is to be moved from the stack, where it was passed as a parameter in the USR call in line 2810, and puts this location onto the zero page, in memory locations $CB and $CC (low, high bytes, respectively). Then the ROM location of the character set, $E000, is placed onto zero page, in memory locations $CD and $CE. The X register acts as a page counter (remember: we need to transfer 4 pages). The Y register counts from 0 back to 0, transferring the whole page. The loop from line 440 to line 520 transfers all 4 pages, with the loop from 450 to 480 transferring each page. The transfer uses the Y register as an offset into the ROM and RAM versions of the character set, from the base addresses, stored on the zero page.

After each page is transferred, the base address of each version (ROM and RAM) is incremented in lines 490 and 500, and the loop repeated.

By way of comparison, this routine in BASIC takes between 20 and 30 seconds to execute, depending on positioning within the program, and other factors. This machine language subroutine takes 14433 machine cycles to execute. The cycle time of the ATARI 6502A processor is 560 nanoseconds, so the total time of execution is 8082 microseconds, or about eight one-thousanths of a second! That's the advantage of machine language over BASIC.

n. Line 2900

Before we play the music, we set up the screen, and make the cursor go away (POKE 752,1). We then set the pitch of the various notes to be used, in lines 2940 and 2950. The notes are then played in the following lines. Note that the speed of the music increases as it proceeds, because the markers WH, HA and QT (WHole note, HAlf note, and QuarTer note) for the DURation, are made smaller in lines 3120, 3230, 3270 and 3290. Note that at intervals as the notes are played, the program PRINTs a dot on the screen in a random location, giving the appearance of a star field becoming visible.

The sharp-eyed reader of the source code will see a number of lines that are considerably longer than the 3 physical lines allowed by the BASIC cartridge. How is this possible? By some more trickery. Note, for example, line 3080, or line 3130. These lines have over 160 characters in them! Two different methods are available for your ATARI to achieve this. The first, to abbreviate all BASIC commands as you type in the line, is commonly used, but you can see that this won't allow those two lines to be that long. The second method was used for these. If you LIST a BASIC program to disk or tape, you can ENTER it into the computer with the Assembler/Editor cartridge in place. The Editor will allow you to replace any string with any other, so that you will be able to put in some code of your choice, like XXXXXX, and then replace it with a whole line of additional BASIC throughout your program. One caution should be exercised, however. This line will be accepted by the BASIC cartridge when you reENTER the program into BASIC, but don't try to edit this line, or you'll lose it. The real usefulness of this trick is to cram as much as possible on each line, which will speed program execution and save memory as well (fewer line numbers for the computer to store).

o. Line 3420

Instructions to a program can either be provided in printed form, or can reside in the computer itself, within the program. In ASTEROID MINERS, the second option is used, and this subroutine prints these to the screen. Note particularly that when the screen is full, the program uses a GET to allow the users to read that full screen of instructions at their own pace, and then, by hitting any key, allow the next screenful to appear. This is particularly helpful to beginners, who may be uncomfortable with using the computer in the first place. Note also the use of the dark red letters on the light pink background, to avoid monotony in the appearance of the program.

p. Line 3750

These routines are very straightforward, and simply take the numbers resulting from the bit maps of the shapes of the players and POKE them into protected memory. There is one trick here, however. We've run out of protected memory. How to get more? We could lower MEMTOP again, and store the shapes there, but we've already done that twice. There is one other place usually used, page 6, but we're going to store our machine language routines there. Well, there's still one other place, but it requires some care.

When your ATARI GOSUBs or executes a FOR...NEXT loop, have you ever wondered how it knows where it was, so that it can properly RETURN or jump back to the FOR statement? The answer is that it stores the address it has just been at on the stack, which, for the 6502A, is page 1 of memory. This acts just like a stack of dishes on a spring loaded pile, as is common in cafeterias. As we add a plate to the pile, all the other plates move down one position, and when we remove a plate from the top, all the others move up. The return addresses act exactly the same way, and when the 6502A encounters a RETURN, it pulls the top address off the stack and goes there. So how does all of this help us?

The stack is a full page of memory, 256 bytes. If you're careful with your nested loops, you won't use nearly that many locations. Therefore, the bottom of the stack, $0100 and up, is available for you on which to store anything. Hexidecimal $0100 is decimal 256, so we start storing the shapes at 260, just to keep things even. You may have noticed that the shapes of the mushroom cloud in the explosion subroutine are also located on page 1. Just remember, don't use any locations much above 490, or you could run into trouble, but this number will vary greatly from program to program. You'll just have to experiment if you need the storage space. Page 1 ends at memory location 511.

q. Line 3940

Since the three remaining machine language subroutines are located together at the end of the program, they will be discussed as if they were three subroutines, although all the lines from 3940 on accomplish is to POKE them into memory, not execute them. The lines from 3940 on are really called in two sections: 1. from the GOSUB 3940 in line 1110, and 2. from the GOSUB 4010 in line 1140. They are called this way because they take too much time to execute together (POKEing is a slow business).

1. Moving the Players

This machine language routine is short and fast. The first PLA is needed for BASIC to access the subroutine. The next four lines pull the location of the player off the stack, and store it on page zero. Then we pull the pointer to the shape we want to draw at that location, and also store that on page zero. The loop from 410-450 moves 8 bytes from where the shapes are stored to where we want them to appear, and we return to BASIC. Short, sweet, and very fast. It uses only 182 machine cycles, or about 0.1 milliseconds!

2. Clearing Memory to all Zeros

The use of this routine in ASTEROID MINERS is to clear out the space used for the player-missle graphics, but it could be adapted for any other use, as well. Again we start with the PLA BASIC needs. Lines 330-430 set up the low byte of the indirect page zero address as 0, and the high byte is 7 less than the number stored in $6A (decimal 106), the pointer the ATARI uses to point to the top of memory. The loop from 440-510 clears all these pages, with the loop from 450-470 clearing each page. Remember, line 330 loaded the accumulator with 0, so we don't have to load it with 0 again in line 450, or ever again, in fact, as long as we use the other registers as counters. Note that counting the Y register down from $FF to 0 allows us to branch without a compare, since the 6502A sets a flag when such operations reach zero, and we can simply test for the equality to zero, as in line 470.

3. The Countdown Timer

This is the longest and most complex machine language subroutine in ASTEROID MINERS, and is extensively modified from a routine for a real-time clock originally published by Craig Patchett in A.N.A.L.O.G. 400/800, Issue #5, 1981. The 3 digits of the timer have locations $0650-$0652 reserved for them in line 490. INDIC, location $0653, will be used to signal that time has expired, so we know when to end the game. INDIC1, location $0654, will be used to signal that the minutes digit has changed. Remember, we use that to add a random asteroid to the field at the highest level of difficulty. Finally, TIMER, location $0655, will be used to actually countdown from 60 to 0, since the vertical blank interrupt occurs 60 times per second. Each time this reaches zero, one second has expired.

The routine from lines 630-680 actually inserts this whole routine in the vertical blank interrupt. It does this by stealing the vector the ATARI uses to jump to the proper routine, and sends it to this one first. At the end of our program, line 1340, we then jump to the right place, so we've just inserted our routine in between real time processing, and that processing occurring while the electron beam is waiting to paint another picture on your TV tube, sixty times per second. Time stolen here for our routine is time that BASIC is inoperable, so that this routine actually is simultaneously processed along with ASTEROID MINERS. We have created a multiprocessing computer out of your ATARI! The inserting routine just consists of loading the Y register with the low byte of the address of the routine to be inserted, the X register with the high byte of this address, and the accumulator with a 6 for the first part of the vertical blank interrupt, or a 7 for the second part. We use a 6 here, but for longer routines, 7 has more slack time associated with it. We then JSR to SETVBV, a routine the good folks at ATARI built into your computer's operating system just to change the vertical blank interrupt vectors. That's all there is to it! Next, we'll examine the routine for the timer itself.

Lines 760-810 check to see if TIMER is zero. If not, they print out the digits. If it is, lines 820-830 reset the TIMER, and then lines 840-1110 check each of the digits. First the lowest order digit is decreased by one. If it is now less than zero (=$FF), then it is set to the right digit, and we proceed to the next higher digit. The right digit in this case is a 9 for the seconds, and a 5 for the tens of seconds. In any case, each subpart for each digit stores the correct digit in the storage locations reserved for them, as discussed above. Lines 1010-1020 set the flag INDIC1, and lines 1120-1130 set INDIC, both of which were also discussed above.

We want the actual digits of the timer to appear in the middle of the fifth line of a GRAPHICS 0 screen, so the appropriate offset to produce this was calculated to be 179, and is used in the printing routine, from line 1140-1330. Four lines of 40 bytes per line is 160, plus half of one more line minus one to center it, is 179. To print the digits to the screen, we actually POKE them directly into screen memory. First the minutes digit, in lines 1260-1320, then a colon, in lines 1180-1250, then the next two digits using the same routine in lines 1260-1320. This finishes the routine, so we exit by jumping to the normal exit for the first part of the vertical blank interrupt routine, SYSVBV.

The routine in lines 1440-1490 simply reroute the vertical blank interrupt routine back to the way it way initially. Since the timer routine will not be accessed then, the timer effectively shuts off until we reinsert the routine. This allows us to turn the timer on and off at will!

Conclusion

That completes the explanation of ASTEROID MINERS. We hope that you have learned a great deal about your ATARI from this tutorial. In turn, we hope to learn a great deal from you. What do you think of this approach? Would you like to see more of it? How would you improve it? Any and all suggestions you have for improving or extending this will be read and considered in depth, and we would very much like to hear from you. Thank you in advance for your suggestions, criticisms, and help. And enjoy ASTEROID MINERS, both the game and the tutorial!

ASTEROID MINERS

A UNIQUE GAME TUTORIAL

by Mark Chasin

The Complete,

Documented and

Commented

BASIC Program Listing

```
100 POKE 77,0
110 Q0=0:Q1=1:Q2=2:Q3=3:Q4=4:Q5=5:Q6=6:Q7=7:
Q8=8:Q9=9:Q10=10:Q12=12:Q14=14:Q15=15:Q16=16
:Q17=0.5:Q19=19:Q21=20:Q34=34:Q40=40
120 Q41=2040:Q49=49:Q64=64:Q96=96:Q97=97:Q10
0=100:Q256=256:Q512=512:Q536=1536:Q559=559:Q
720=720:Q752=752:Q774=53774
130 POKE 580,Q1:POKE Q559,Q0:GOTO 1000
140 REM
150 REM
160 REM ******************************
170 REM ******************************
180 REM ***      ASTEROID MINERS     ***
190 REM *** A UNIQUE GAME TUTORIAL ***
200 REM ***     (c)1982 MMG MICRO    ***
210 REM ***         SOFTWARE         ***
220 REM ***                          ***
230 REM ***      by Mark Chasin      ***
240 REM ***                          ***
250 REM ******************************
260 REM ******************************
270 REM
280 REM
290 REM
300 REM
310 REM THIS SUBROUTINE DETERMINES THE POSIT
ION OF THE JOYSTICK AND LIMITS THE PLAYERS T
O THE SIZE OF THE SCREEN
320 REM
330 REM
340 REM
350 IF (J=11) OR (J=Q10) OR (J=Q9) THEN X=X-
MOV
360 IF (J=Q6) OR (J=Q7) OR (J=Q5) THEN X=X+M
OV
370 IF (J=Q10) OR (J=Q14) OR (J=Q6) THEN Y=Y
-MOV
380 IF (J=Q9) OR (J=13) OR (J=Q5) THEN Y=Y+M
OV
390 IF Y>Q100 THEN Y=Q100
400 IF Y<Q40 THEN Y=Q40
410 IF X<Q49 THEN X=Q49
420 IF X>190 THEN X=190
430 RETURN
440 REM
450 REM
460 REM
470 REM THIS SUBROUTINE CHANGES THE DIRECTIO
```

```
N THE PLAYERS ARE FACING, AND ADJUSTS THEIR
NEW POSITIONS
480 REM
490 REM
500 REM
510 IF J=Q14 THEN FACING=Q1:Y=Y-MOV:GOTO 630
520 IF J=Q6 THEN FACING=Q2:Y=Y-MOV:X=X+MOV:G
OTO 630
530 IF J=Q7 THEN FACING=Q3:X=X+MOV:GOTO 630
540 IF J=Q5 THEN FACING=Q4:X=X+MOV:Y=Y+MOV:G
OTO 630
550 IF J=13 THEN FACING=Q5:Y=Y+MOV:GOTO 630
560 IF J=Q9 THEN FACING=Q6:Y=Y+MOV:X=X-MOV:G
OTO 630
570 IF J=11 THEN FACING=Q7:X=X-MOV:GOTO 630
580 IF J=Q10 THEN FACING=Q8:X=X-MOV:Y=Y-MOV
590 IF Y>Q100 THEN Y=Q100
600 IF Y<Q40 THEN Y=Q40
610 IF X<Q49 THEN X=Q49
620 IF X>190 THEN X=190
630 RETURN
640 REM
650 REM
660 REM
670 REM THIS SUBROUTINE DETERMINES THE DEGRE
E OF DIFFICULTY AND SETS UP THE APPROPRIATE
PARAMETERS
680 REM
690 REM
700 REM
710 DIM VOLUE(Q3,Q3)
720 GRAPHICS 18:SETCOLOR Q4,Q0,Q0:? #Q6:? #Q
6:? #Q6:? #Q6;"         degree":? #Q6;"   of d
ifficulty":? #Q6:? #Q6;"         (1-3)";
730 REM POKE 16,64:POKE 53774,64
740 OPEN #Q1,Q4,Q0,"K:":GET #Q1,ANSWER:CLOSE
 #Q1:IF (ANSWER<Q49) OR (ANSWER>51) THEN 720
750 VOLUE(Q1,Q1)=-Q10:VOLUE(Q1,Q2)=-Q100:VOL
UE(Q1,Q3)=-400:VOLUE(Q2,Q1)=Q10:VOLUE(Q2,Q2)
=50
760 VOLUE(Q2,Q3)=200:VOLUE(Q3,Q1)=30:VOLUE(Q
3,Q2)=150:VOLUE(Q3,Q3)=600
770 IF ANSWER=Q49 THEN SPLIM=Q15:ALIM=Q8:BLI
M=Q6:DIFF=Q1:TIME=Q5
780 IF ANSWER=50 THEN SPLIM=Q21:ALIM=Q10:BLI
M=Q7:DIFF=Q2:TIME=Q7
790 IF ANSWER=51 THEN SPLIM=28:ALIM=Q15:BLIM
=Q12:DIFF=Q3:TIME=Q9
```

```
800 ? #Q6;? #Q6;? #Q6;? #Q6;"     GET READY!
";:IF QIF>Q1 THEN FOR I=Q1 TO 400:NEXT I
810 REM POKE 16,64:POKE 53774,64
820 RETURN
830 REM
840 REM
850 REM
860 REM FRONTPIECE OF THE GAME
870 REM
880 REM
890 REM
900 GRAPHICS Q2:SETCOLOR Q4,Q0,Q0:SETCOLOR Q
2,Q0,Q0:POKE Q752,Q1:SETCOLOR Q1,Q0,Q14:? #Q
6:? #Q6;"   asteroid miners"
910 ? "      (c)1982 MMG Micro Software":? "
          by Mark Chasin"
920 RETURN
930 REM
940 REM
950 REM
960 REM SET UP THE MAIN GAME
970 REM
980 REM
990 REM
1000 GOSUB 2800:GOSUB 2900
1010 ? "   Do you need instructions (Y/N)?"
;:OPEN #Q1,Q4,Q0,"K:":GET #Q1,ANS:CLOSE #Q1:
IF (ANS<>89) AND (ANS<>78) THEN 1010
1020 REM POKE 16,64:POKE 53774,64
1030 IF ANS=89 THEN GOSUB 3420
1040 ? "":SOUND Q0,121,Q12,Q4:SOUND Q1,140,
Q8,Q4:SETCOLOR Q4,Q0,Q0:POSITION Q0,Q4:? #Q6
;"  YOU ARE ABOUT TO"
1050 ? #Q6;"  EMBARK ON A TRIP":? #Q6;"  THR
OUGH TIME AND":? #Q6;"       SPACE.":GOSUB 3
750
1060 REM POKE 16,64:POKE 53774,64
1070 POSITION Q0,Q3:? #Q6;" A TREASURE AWAIT
S":? #Q6;" THE ADVENTURESOME ":? #Q6;" IN TH
E DEPTHS OF   "
1080 ? #Q6;"  SPACE, BUT FOR   ":? #Q6;"THE
CARELESS AWAITS";:FOR BBB=Q1 TO 600:NEXT BBB
:? #Q6:? #Q6;"        death";
1090 REM POKE 16,64:POKE 53774,64
1100 SOUND Q2,85,Q10,Q14:SOUND Q3,Q96,Q10,Q1
4:FOR NBC=Q1 TO Q40:NEXT NBC:SOUND Q2,Q0,Q0,
Q0:SOUND Q3,Q0,Q0,Q0:FOR Q=Q1 TO Q5:NEXT Q
1110 SOUND Q2,91,Q10,Q14:SOUND Q3,108,Q10,Q1
```

```
4:FOR NBC=Q1 TO 500:NEXT NBC:SOUND Q2,Q0,Q0,
Q0:SOUND Q3,Q0,Q0,Q0:GOSUB 3940
1120 POSITION Q0,Q3:? #Q6;"  YOUR JOURNEY IS
 ":? #Q6;" OF CRUCIAL IMPOR- ":? #Q6;"TANCE
TO EARTH, BUT"
1130 ? #Q6;" WE KNOW THAT YOU, ":? #Q6;"THE
BEST OF ALL OF ":? #Q6;" US, WILL SUCCEED. "
1140 ? #Q6;"     good luck    ":GOSUB 4010
1150 REM POKE 16,64:POKE 53774,64
1160 GOSUB 710:FOR N=Q15 TO 0 STEP -Q17:SOUN
D Q0,Q21,Q10,N:NEXT N:SOUND Q0,Q0,Q0,Q0:GOSU
B 2030
1170 A=PEEK(106)-Q8:POKE 54279,A:POKE 53277,
Q2:PMBASE=Q256*A:POKE 53256,Q1:X=124:Y=48:FA
CING=Q5
1180 POKE 704,116:POKE 705,68:POKE 706,Q0:FO
R N=Q15 TO 0 STEP -Q17:SOUND Q0,Q21,Q10,N:NE
XT N:SOUND Q0,Q0,Q0,Q0:GOSUB 2140
1190 D=USR(1568):GRAPHICS Q0:SETCOLOR Q1,Q0,
Q0:SETCOLOR Q2,Q0,Q0:POKE 708,Q0:POKE Q752,Q
1:POKE 756,NOWCSET
1200 REM
1210 POSITION Q14,Q0:? "ASTEROID MINERS";:PO
SITION Q0,Q2:? "SCORE";:POSITION Q19,Q2:? "T
IME";:POSITION Q34,Q2:? "HIGH";
1220 REM POKE 16,64:POKE 53774,64
1230 POSITION Q0,Q3:? "_____";:POSITION Q19,
Q3:? "____";:POSITION Q34,Q3:? "____";
1240 FOR N=Q15 TO 0 STEP -Q17:SOUND Q0,Q21,Q
10,N:NEXT N:SOUND Q0,Q0,Q0,Q0:GOSUB 1820
1250 FOR N=Q15 TO 0 STEP -Q17:SOUND Q0,Q21,Q
10,N:NEXT N:SOUND Q0,Q0,Q0,Q0:POKE 53257,Q1:
POKE 53258,Q1
1260 POSITION Q34,Q4:? HISCORE
1270 C=Q0:FOR A=PMBASE+Q512+Y TO PMBASE+519+
Y:POKE A,PEEK(PTR(FACING)+C):C=C+Q1:NEXT A:P
OKE Q559,42:POKE 623,Q1
1280 SETCOLOR Q1,Q0,Q14:FOR I=1617 TO 1620:P
OKE I,Q0:NEXT I:POKE 1621,Q96:POKE 1616,TIME
:D=USR(1622)
1290 REM
1300 REM
1310 REM
1320 REM ****** THIS IS THE MAIN LOOP ******
1330 REM
1340 REM
1350 REM
1360 BB=Y:IF PEEK(1619) THEN 2250
```

```
1370 CC=X:POKE 53278,Q1:POKE 53248,X:POKE 53
249,X:POKE 53250,X:JJ=J:REM POKE 16,64:POKE
53774,64
1380 J=STICK(Q0):BUT=STRIG(Q0):MOV=Q1:IF BUT
=Q0 THEN MOV=Q4
1390 IF JJ=J THEN GOSUB 350
1400 IF J=Q15 THEN FACINGFL=Q9:SOUND Q0,Q0,Q
0,Q0:SOUND Q1,Q0,Q0,Q0:D=USR(Q536,PMBASE+640
+Y,PTRFL(FACINGFL)):GOTO 1360
1410 POKE 77,Q0:IF J<>JJ THEN GOSUB 510
1420 FACINGFL=FACING:SOUND Q0,240,Q6,Q10:SOU
ND Q1,230,Q8,Q10:D=USR(Q536,PMBASE+Q512+BB,P
TR(Q9))
1430 D=USR(Q536,PMBASE+640+BB,PTRFL(Q9)):D=U
SR(Q536,PMBASE+768+BB,PTRIN(Q9))
1440 D=USR(Q536,PMBASE+Q512+Y,PTR(FACING)):D
=USR(Q536,PMBASE+640+Y,PTRFL(FACINGFL))
1450 D=USR(Q536,PMBASE+768+Y,PTRIN(FACING)):
IF (DIFF=Q3) AND (PEEK(1620)) THEN GOSUB 242
0:POKE 1620,Q0
1460 P1=PEEK(53252):P2=PEEK(53254)
1470 IF (P1>Q0) AND (DIFF>Q1) THEN GOTO 2640
1480 IF P2=Q0 THEN GOTO 1360
1490 XDIF=Q2*((CC<X)-(X<CC)):MIN=(BB<Y)-(Y<B
B):XXX=INT((X-42)/Q4)+XDIF:YYY=INT((Y-Q14)/3
.8333)+MIN
1500 LOCATE XXX,YYY,VQ:VQVAL=(VQ=99)+Q2*(VQ=
98)+Q3*(VQ=Q97)
1510 IF VQ=32 THEN GOTO 1360
1520 VOLVQ=VOLUE(VQVAL,DIFF):IF VOLVQ=Q1 THE
N VOLVQ=Q0
1530 POSITION XXX,YYY:? " ";:IF VOLVQ>Q1 THE
N FOR MX=250 TO 115 STEP -Q15:SOUND Q3,MX,Q1
0,Q10:NEXT MX:SOUND Q3,Q0,Q0,Q0
1540 IF VOLVQ>=Q0 THEN 1560
1550 FOR MX=150 TO Q15 STEP -Q15:SOUND Q3,MX
,Q8,Q12:NEXT MX:FOR MX=Q15 TO 150 STEP Q15:S
OUND Q3,MX,Q8,Q12:NEXT MX:SOUND Q3,Q0,Q0,Q0
1560 IF VOLVQ>Q1 THEN NUMGOOD=NUMGOOD-Q1
1570 SCORE=SCORE+VOLVQ:POSITION Q0,Q4:? SCOR
E;"   ";:IF NUMGOOD<>Q0 THEN GOTO 1360
1580 D=USR(1762):GOSUB 2510:POKE 53248,Q0:PO
KE 53249,Q0:POKE 53250,Q0:GOSUB 1820:Y=48:X=
124:A=PMBASE+Q512+BB
1590 D=USR(Q536,A,PTR(Q9)):D=USR(Q536,A+128,
PTRFL(Q9)):D=USR(Q536,A+Q256,PTRIN(Q9)):A=A-
Q8
1600 D=USR(Q536,A,PTR(Q9)):D=USR(Q536,A+128,
PTRFL(Q9)):D=USR(Q536,A+Q256,PTRIN(Q9)):A=A+
Q16
1610 D=USR(Q536,A,PTR(Q9)):D=USR(Q536,A+128,
PTRFL(Q9)):D=USR(Q536,A+Q256,PTRIN(Q9))
1620 D=USR(Q536,A,PTR(Q9)):D=USR(Q536,A+128,
PTRFL(Q9)):D=USR(Q536,A+Q256,PTRIN(Q9))
1630 BONUS=DIFF*DIFF*200:POSITION Q0,Q1:? "B
ONUS";:INCRE=50+50*(DIFF=Q3)
1640 FOR DD=INCRE TO BONUS STEP INCRE:POSITI
ON Q0,Q4:SCORE=SCORE+INCRE:? SCORE;"    ";
1650 FOR N=Q15 TO 0 STEP -Q17:SOUND Q0,Q21,Q
10,N:NEXT N:SOUND Q0,Q0,Q0,Q0:NEXT DD:POSITI
ON Q0,Q1:? "         ";
1660 C=Q0:FOR A=PMBASE+Q512+Y TO PMBASE+519+
Y:POKE A,PEEK(PTR(Q5)+C):C=C+Q1:NEXT A:POKE
Q559,42:POKE 623,Q1
1670 D=USR(1622)
1680 GOTO 1360
1690 REM
1700 REM
1710 REM
1720 REM ******  END OF THE MAIN LOOP ******
1730 REM
1740 REM
1750 REM
1760 REM
1770 REM
1780 REM THIS SECTION SETS UP THE FIELD OF A
STEROIDS, USING THE REDEFINED CHARACTER SET
1790 REM
1800 REM
1810 REM
1820 FOR I=Q1 TO SPLIM
1830 XX=INT(RND(Q0)*38):YY=INT(RND(Q0)*Q15+Q
7):LOCATE XX,YY,ZZ:IF ZZ=Q97 THEN 1830
1840 POSITION XX,YY:? "c";:NEXT I:REM POKE 1
6,64:POKE 53774,64
1850 CTA=Q0:FOR I=Q1 TO ALIM
1860 XX=INT(RND(Q0)*36+Q1):YY=INT(RND(Q0)*Q1
5+Q7):LOCATE XX,YY,ZZ:IF (ZZ=Q97) OR (ZZ=98)
 THEN 1860
1870 IF ((YYY=Q8) OR (YYY=Q9)) AND ((XXX>=18
) AND (XXX<=23)) THEN 1860
1880 POSITION XX,YY:? "b";:CTA=CTA+Q1:NEXT I
:REM POKE 16,64:POKE 53774,64
1890 FOR I=Q1 TO BLIM
1900 XX=INT(RND(Q0)*36+Q1):YY=INT(RND(Q0)*Q1
5+Q7):LOCATE XX,YY,ZZ:IF (ZZ>Q96) AND (ZZ<Q1
```

```
00) THEN 1900
1910 IF ((YYY=Q8) OR (YYY=Q9)) AND ((XXX>=18
) AND (XXX<=23)) THEN 1900
1920 POSITION XX,YY:? "a";:CTA=CTA+Q1:NEXT I
:NUMGOOD=CTA:REM POKE 16,64:POKE 53774,64
1930 FOR VY=Q8 TO Q9:FOR VX=18 TO 23:LOCATE
VX,VY,VZ:IF (VZ=Q97) OR (VZ=98) THEN NUMGOOD
=NUMGOOD-Q1
1940 NEXT VX:NEXT VY:FOR ABC=Q8 TO Q9:POSITI
ON Q19,ABC:? "      ";:NEXT ABC
1950 RETURN
1960 REM
1970 REM
1980 REM
1990 REM THIS SUBROUTINE REDEFINES THE CHARA
CTER SET IN RAM AS WE NEED IT
2000 REM
2010 REM
2020 REM
2030 RESTORE 2040:FOR I=Q1 TO Q4:READ RPLC:F
OR J=Q0 TO Q7:READ AB:POKE RAM+Q8*RPLC+J,AB:
NEXT J:NEXT I
2040 DATA 97,8,42,170,170,170,170,42,8,98,16
,84,85,85,85,85,84,4,99,24,124,126,255,255,1
26,126,24
2050 DATA 13,0,0,0,126,126,0,0,0
2060 RETURN
2070 REM
2080 REM
2090 REM
2100 REM THIS SUBROUTINE SETS UP THE POINTER
S TO THE LOCATIONS OF EACH OF THE SHAPES OF
THE PLAYERS
2110 REM
2120 REM
2130 REM
2140 RESTORE 2140:DIM PTR(Q9):FOR A=Q1 TO Q9
:READ I:PTR(A)=I:NEXT A:DATA 260,268,276,284
,292,300,308,316,324
2150 DIM PTRPL(Q9):FOR A=Q1 TO Q9:READ I:PTR
PL(A)=I:NEXT A:DATA 332,340,348,356,364,372,
380,388,396
2160 DIM PTRIN(Q9):FOR A=Q1 TO Q9:READ I:PTR
IN(A)=I:NEXT A:DATA 404,412,420,428,436,444,
452,460,468
2170 RETURN
2180 REM
2190 REM
```

```
2200 REM
2210 REM THIS SUBROUTINE RECONFIGURES THE SC
REEN AFTER THE GAME ENDS
2220 REM
2230 REM
2240 REM
2250 SOUND Q0,Q0,Q0,Q0:SOUND Q1,Q0,Q0,Q0:IF
SCORE>HISCORE THEN HISCORE=SCORE
2260 D=USR(1762)
2270 POSITION Q14,Q0:? "ASTEROID MINERS";:PO
SITION Q0,Q2:? "SCORE";:POSITION Q19,Q2:? "T
IME";:POSITION Q34,Q2:? "HIGH";
2280 POSITION Q0,Q3:? "_____";:POSITION Q19,
Q3:? "____";:POSITION Q34,Q3:? "____";:POSIT
ION Q34,Q4:? HISCORE
2290 REM POKE 16,64:POKE 53774,64
2300 POKE 53248,Q0:POKE 53249,Q0:POKE 53250,
Q0
2310 FOR I=Q5 TO 23:POSITION Q0,Q5:? "";:NE
XT I:SCORE=Q0:OPEN #Q1,Q4,Q0,"K:":SETCOLOR Q
2,Q0,Q0:SETCOLOR Q1,Q0,Q14
2320 POSITION Q10,13:? "ANOTHER GAME: Y OR N
?":GET #Q1,ANS:CLOSE #Q1:POSITION Q0,13:? "
";:IF ANS<>89 THEN GRAPHICS Q0:NEW
2330 POKE Q16,Q64:POKE Q774,Q64:GOSUB 720:FA
CING=Q5:SCORE=Q0:Y=48:X=124:POSITION Q0,Q4:?
 "     ";
2340 GOTO 1190
2350 REM
2360 REM
2370 REM
2380 REM THIS SUBROUTINE ADDS A RANDOM ASTER
OID IN THE HIGHEST LEVEL OF DIFFICULTY
2390 REM
2400 REM
2410 REM
2420 XX=INT(RND(Q0)*36+Q1):YY=INT(RND(Q0)*Q1
5+Q7):LOCATE XX,YY,ZZ:IF (ZZ>Q96) AND (ZZ<Q1
00) THEN 2420
2430 POSITION XX,YY:? CHR$(Q97+INT(RND(Q0)+Q
17));:NUMGOOD=NUMGOOD+Q1:RETURN
2440 REM
2450 REM
2460 REM
2470 REM THE SOUND YOU HEAR IS THE HERALDING
 OF A SUCCESSFUL CLEARING OF A FULL SCREEN!!
!
2480 REM
```

```
2490 REM
2500 REM
2510 SOUND Q0,Q0,Q0,Q0:SOUND Q1,Q0,Q0,Q0:SOU
ND Q3,121,Q10,Q8:SOUND Q2,144,Q10,Q8:FOR I=Q
1 TO Q10:NEXT I
2520 SOUND Q2,Q0,Q0,Q0:SOUND Q3,Q0,Q0,Q0:SOU
ND Q3,91,Q10,Q8:SOUND Q2,108,Q10,Q8:FOR I=Q1
 TO Q21:NEXT I
2530 SOUND Q2,Q0,Q0,Q0:SOUND Q3,Q0,Q0,Q0:SOU
ND Q3,121,Q10,Q8:SOUND Q2,144,Q10,Q8:FOR I=Q
1 TO Q10:NEXT I
2540 SOUND Q2,Q0,Q0,Q0:SOUND Q3,Q0,Q0,Q0:SOU
ND Q3,91,Q10,Q8:SOUND Q2,108,Q10,Q8:FOR I=Q1
 TO Q40:NEXT I
2550 SOUND Q2,Q0,Q0,Q0:SOUND Q3,Q0,Q0,Q0
2560 RETURN
2570 REM
2580 REM
2590 REM
2600 REM SUBROUTINE-YOU HIT AN ASTEROID; A D
EFINITE NO-NO!!
2610 REM
2620 REM
2630 REM
2640 POKE 53248,Q0:POKE 53249,Q0:POKE 53250,
Q0:POKE 53259,Q0:AB=22:POKE 707,AB:POKE 480,
24:POKE 487,24
2650 POKE 481,60:POKE 486,60:POKE 482,126:PO
KE 485,126:POKE 483,255:POKE 484,255
2660 C=Q0:FOR A=PMBASE+896+Y TO PMBASE+903+Y
:POKE A,PEEK(480+C):C=C+Q1:NEXT A:POKE Q559,
42:POKE 623,Q1:POKE 53251,X
2670 FOR I=Q1 TO 45:IF I/Q5=INT(I/Q5) THEN A
B=AB+Q16:POKE 707,AB
2680 SOUND Q0,RND(Q0)*Q256,Q8,Q8:IF I=Q15 TH
EN AB=22:POKE 707,AB:POKE 53251,X-Q5:POKE 53
259,Q1
2690 IF I=30 THEN AB=22:POKE 707,AB:POKE 532
51,X-Q10:POKE 53259,Q3
2700 SOUND Q1,RND(Q0)*Q256,Q6,Q8:SOUND Q2,RN
D(Q0)*Q256,Q12,Q8:SOUND Q3,RND(Q0)*Q256,Q8,Q
8:SETCOLOR Q2,RND(Q0)*Q16,Q8
2710 NEXT I:SETCOLOR Q2,Q4,Q10:SETCOLOR Q1,Q
4,Q10:FOR I=Q0 TO Q3:SOUND I,Q0,Q0,Q0:NEXT I
:POKE 53251,Q0
2720 GOTO 2250
2730 REM
2740 REM
```

```
2750 REM
2760 REM SUBROUTINE TO MOVE THE CHARACTER SE
T FROM ROM TO RAM
2770 REM
2780 REM
2790 REM
2800 POKE Q559,Q0:ROM=57344:RAM=(PEEK(106)-Q
4)*Q256:NOWCSET=PEEK(106)-Q4:POKE 106,NOWCSE
T:POKE 756,NOWCSET
2810 DIM QWER$(Q34):QWER$="hhLhK)M)`N"
 1MKHPyfLfNJPp`":A=USR(ADR(QWER$),RAM)
2820 RETURN
2830 REM
2840 REM
2850 REM
2860 REM SUBROUTINE TO PLAY 2001 - A SPACE O
DYSSEY, AND THEN SET UP THE SCREEN
2870 REM
2880 REM
2890 REM
2900 GRAPHICS Q2:SETCOLOR Q4,Q0,Q0:SETCOLOR
Q2,Q0,Q0:POKE Q752,Q1:SETCOLOR Q1,Q0,Q14:? #
Q6:? #Q6;"   asteroid miners"
2910 C=243:F=182:G=162:A=144:AS=136:? "
(c)1982 MMG Micro Software":? "          by
 Mark Chasin"
2920 REM POKE 16,64:POKE 53774,64
2930 C1=121:D1=108:E1=Q96:F1=91:G1=81:G1S=76
:A1=72:A1S=68:XV=INT(RND(Q0)*Q21):YV=INT(RND
(Q0)*Q8+Q2):POSITION XV,YV:? #Q6;".";
2940 C2=60:D2=53:E2=47:F2=45:G2=Q40:A2=35:A2
S=33:C3=29:WH=70:HF=35:QT=Q21:ET=Q10:SX=Q5
2950 X=Q10:Y=Q15:XV=INT(RND(Q0)*Q21):YV=INT(
RND(Q0)*Q8+Q2):POSITION XV,YV:? #Q6;".";
2960 FOR R=Q0 TO Q3:SOUND R,F,X,Y:NEXT R:FOR
 DUR=Q1 TO WH*Q2:NEXT DUR
2970 FOR R=Q0 TO Q3:SOUND R,C1,X,Y:NEXT R:FO
R DUR=Q1 TO WH*Q2:NEXT DUR:XV=INT(RND(Q0)*Q2
1):YV=INT(RND(Q0)*Q8+Q2):POSITION XV,YV:? #Q
6;".";
2980 FOR R=Q0 TO Q3:SOUND R,F1,X,Y:NEXT R:FO
R DUR=Q1 TO (WH+HF+QT)*Q2:NEXT DUR:XV=INT(RN
D(Q0)*Q21):YV=INT(RND(Q0)*Q8+Q2):POSITION XV
,YV:? #Q6;".";
2990 FOR R=Q0 TO Q3:SOUND R,A1,X,Y:NEXT R:FO
R DUR=Q1 TO QT:NEXT DUR:XV=INT(RND(Q0)*Q21):
YV=INT(RND(Q0)*Q8+Q2):POSITION XV,YV:? #Q6;"
.";
```

```
3000 SOUND Q0,F,X,Y:SOUND Q1,C1,X,Y:SOUND Q2
,F1,X,Y:SOUND Q3,G1S,X,Y:FOR DUR=Q1 TO HF:NE
XT DUR
3010 FOR QQ=Q1 TO Q3:SOUND Q0,C,X,Y:FOR DUR=
Q1 TO HF:NEXT DUR:SOUND Q0,F,X,Y:FOR DUR=Q1
TO HF:NEXT DUR:NEXT QQ:XV=INT(RND(Q0)*Q21):Y
V=INT(RND(Q0)*Q8+Q2):POSITION XV,YV:? #Q6;".
";
3020 SOUND Q0,C,X,Y:FOR DUR=Q1 TO HF:NEXT DU
R:XV=INT(RND(Q0)*Q21):YV=INT(RND(Q0)*Q8+Q2):
POSITION XV,YV:? #Q6;".";
3030 FOR R=Q0 TO Q3:SOUND R,F,X,Y:NEXT R:FOR
 DUR=Q1 TO WH:NEXT DUR:XV=INT(RND(Q0)*Q21):Y
V=INT(RND(Q0)*Q8+Q2):POSITION XV,YV:? #Q6;".
";
3040 FOR R=Q0 TO Q3:SOUND R,C1,X,Y:NEXT R:FO
R DUR=Q1 TO WH:NEXT DUR
3050 FOR R=Q0 TO Q3:SOUND R,F1,X,Y:NEXT R:FO
R DUR=Q1 TO WH+HF+QT:NEXT DUR:XV=INT(RND(Q0)
*Q21):YV=INT(RND(Q0)*Q8+Q2):POSITION XV,YV:?
 #Q6;".";
3060 QT=QT-Q2:XV=INT(RND(Q0)*Q21):YV=INT(RND
(Q0)*Q8+Q2):POSITION XV,YV:? #Q6;".";
3070 FOR R=Q0 TO Q3:SOUND R,G1S,X,Y:NEXT R:F
OR DUR=Q1 TO QT:NEXT DUR
3080 SOUND Q0,F,X,Y:SOUND Q1,C1,X,Y:SOUND Q2
,F1,X,Y:SOUND Q3,A1,X,Y:FOR DUR=Q1 TO HF:NEX
T DUR:XV=INT(RND(Q0)*Q21):YV=INT(RND(Q0)*Q8+
Q2):POSITION XV,YV:? #Q6;".";
3090 FOR QQ=Q1 TO Q3:SOUND Q0,C,X,Y:FOR DUR=
Q1 TO HF:NEXT DUR:SOUND Q0,F,X,Y:FOR DUR=Q1
TO HF:NEXT DUR:NEXT QQ
3100 SOUND Q0,A,X,Y:FOR DUR=Q1 TO HF:NEXT DU
R:XV=INT(RND(Q0)*Q21):YV=INT(RND(Q0)*Q8+Q2):
POSITION XV,YV:? #Q6;".";
3110 SOUND Q0,AS,X,Y:SOUND Q1,F1,X,Y:SOUND Q
2,A1S,X,Y:SOUND Q3,D2,X,Y:FOR DUR=Q1 TO HF:N
EXT DUR
3120 WH=WH-Q4:HF=HF-Q4:XV=INT(RND(Q0)*Q21):Y
V=INT(RND(Q0)*Q8+Q2):POSITION XV,YV:? #Q6;".
";
3130 SOUND Q0,C1,X,Y:SOUND Q1,F1,X,Y:SOUND Q
2,C2,X,Y:SOUND Q3,E2,X,Y:FOR DUR=Q1 TO HF:NE
XT DUR:XV=INT(RND(Q0)*Q21):YV=INT(RND(Q0)*Q8
+Q2):POSITION XV,YV:? #Q6;".";
3140 SOUND Q0,D1,X,Y:SOUND Q1,F1,X,Y:SOUND Q
2,A1S,X,Y:SOUND Q3,F2,X,Y:FOR DUR=Q1 TO WH+H
F+ET:NEXT DUR:XV=INT(RND(Q0)*Q21):YV=INT(RND
(Q0)*Q8+Q2):POSITION XV,YV:? #Q6;".";
3150 SOUND Q0,AS,X,Y:SOUND Q1,F1,X,Y:SOUND Q
2,A1S,X,Y:SOUND Q3,G2,X,Y:FOR DUR=Q1 TO HF+Q
T+ET:NEXT DUR
3160 SOUND Q0,F,X,Y:SOUND Q1,C1,X,Y:SOUND Q2
,F1,X,Y:SOUND Q3,A2,X,Y:FOR DUR=Q1 TO HF:NEX
T DUR
3170 SOUND Q0,G,X,Y:SOUND Q1,C1,X,Y:SOUND Q2
,G1,X,Y:SOUND Q3,AS2,X,Y:FOR DUR=Q1 TO HF:NE
XT DUR:XV=INT(RND(Q0)*Q21):YV=INT(RND(Q0)*Q8
+Q2):POSITION XV,YV:? #Q6;".";
3180 SOUND Q0,A,X,Y:SOUND Q1,C1,X,Y:SOUND Q2
,F1,X,Y:SOUND Q3,C3,X,Y:FOR DUR=Q1 TO HF:NEX
T DUR
3190 SOUND Q2,A1,X,Y:FOR DUR=Q1 TO HF:NEXT D
UR:XV=INT(RND(Q0)*Q21):YV=INT(RND(Q0)*Q8+Q2)
:POSITION XV,YV:? #Q6;".";
3200 SOUND Q2,F1,X,Y:FOR DUR=Q1 TO HF:NEXT D
UR:XV=INT(RND(Q0)*Q21):YV=INT(RND(Q0)*Q8+Q2)
:POSITION XV,YV:? #Q6;".";
3210 SOUND Q2,C1,X,Y:FOR DUR=Q1 TO HF:NEXT D
UR:XV=INT(RND(Q0)*Q21):YV=INT(RND(Q0)*Q8+Q2)
:POSITION XV,YV:? #Q6;".";
3220 SOUND Q2,A,X,Y:FOR DUR=Q1 TO HF:NEXT DU
R
3230 WH=WH-Q3:HF=HF-Q3:QT=QT-Q8:XV=INT(RND(Q
0)*Q21):YV=INT(RND(Q0)*Q8+Q2):POSITION XV,YV
:? #Q6;".";
3240 SOUND Q0,F,X,Y:SOUND Q1,C1,X,Y:SOUND Q2
,F1,X,Y:SOUND Q3,A2,X,Y:FOR DUR=Q1 TO QT:NEX
T DUR:XV=INT(RND(Q0)*Q21):YV=INT(RND(Q0)*Q8+
Q2):POSITION XV,YV:? #Q6;".";
3250 SOUND Q0,G,X,Y:SOUND Q1,C1,X,Y:SOUND Q2
,G1,X,Y:SOUND Q3,AS2,X,Y:FOR DUR=Q1 TO QT:NE
XT DUR:XV=INT(RND(Q0)*Q21):YV=INT(RND(Q0)*Q8
+Q2):POSITION XV,YV:? #Q6;".";
3260 SOUND Q0,A,X,Y:SOUND Q1,C1,X,Y:SOUND Q2
,F1,X,Y:SOUND Q3,C3,X,Y:FOR DUR=Q1 TO WH+HF:
NEXT DUR
3270 WH=WH-Q2:HF=HF-Q2:XV=INT(RND(Q0)*Q21):Y
V=INT(RND(Q0)*Q8+Q2):POSITION XV,YV:? #Q6;".
";
3280 SOUND Q0,G,X,Y:SOUND Q1,S+AS,X,Y:SOUND
Q2,A1S,X,Y:SOUND Q3,D2,X,Y:FOR DUR=Q1 TO WH:
NEXT DUR:XV=INT(RND(Q0)*Q21):YV=INT(RND(Q0)*
Q8+Q2):POSITION XV,YV:? #Q6;".";
3290 WH=WH-Q2:HF=HF-Q2
3300 SOUND Q0,C,X,Y:SOUND Q1,S+AS,X,Y:SOUND
```

```
Q2,G1,X,Y:SOUND Q3,E2,X,Y:FOR DUR=Q1 TO WH:N
EXT DUR:XV=INT(RND(Q0)*Q21):YV=INT(RND(Q0)*Q
8+Q2):POSITION XV,YV:? #Q6;".";
3310 SOUND Q0,F,X,Y:SOUND Q1,A,X,Y:SOUND Q2,
C1,X,Y:SOUND Q3,F2,X,Y:FOR DUR=Q1 TO HF:NEXT
 DUR:XV=INT(RND(Q0)*Q21):YV=INT(RND(Q0)*Q8+Q
2):POSITION XV,YV:? #Q6;".";
3320 FOR QQ=Q1 TO Q4:SOUND Q0,C,X,Y:FOR DUR=
Q1 TO HF:NEXT DUR:SOUND Q0,F,X,Y:FOR DUR=Q1
TO HF:NEXT DUR:NEXT QQ
3330 SOUND Q0,F,X,Y:FOR DUR=Q1 TO Q2*WH:NEXT
 DUR:XV=INT(RND(Q0)*Q21):YV=INT(RND(Q0)*Q8+Q
2):POSITION XV,YV:? #Q6;".";
3340 FOR I=Q0 TO Q3:SOUND I,Q0,Q0,Q0:NEXT I:
RETURN
3350 REM
3360 REM
3370 REM
3380 REM SUBROUTINE TO PRINT THE INSTRUCTION
S TO THE SCREEN, IF ASKED TO
3390 REM
3400 REM
3410 REM
3420 GRAPHICS Q0:SETCOLOR Q2,Q4,Q10:SETCOLOR
 Q1,Q4,Q4:POKE Q752,Q1:REM POKE 16,64:POKE 5
3774,64
3430 ? "Welcome to ASTEROID MINERS.The objec
t of the game is to collect  as much    valu
e as possible, by";
3440 ? " mining":? "the asteroids.  Three di
fferent colorsof asteroids will be in the fi
eld to  be mined.  ";
3450 ? "Their values are:":? :? "COLOR","DIF
FICULTY","VALUE":? "_____","__________","___
__":? " BLUE","     1    ","  10"
3460 ? "     ","      2    ","  50":? "     "
,"      3    "," 200":? :? " GOLD","      1
 ","  30"
3470 ? "     ","      2    "," 150":? "     "
,"      3    "," 600":? :? "WHITE","      1
 "," -10"
3480 ? "     ","      2    ","-100":? "     "
,"      3    ","-400":?
3490 ? "     (hit any key to continue)";:OPEN
 #Q1,Q4,Q0,"K:":GET #Q1,ANS:CLOSE #Q1
3500 GRAPHICS Q0:SETCOLOR Q2,Q4,Q10:SETCOLOR
 Q1,Q4,Q4:POKE Q752,Q1:REM POKE 16,64:POKE 5
3774,64
```

```
3510 ? "  As you can see, try to collect the
  blue and gold asteroids, but avoid thewhit
e ones. Of course, there";
3520 ? " are more";? "of the white ones, so
you'll have to  work at it!"
3530 ? :? "  Your ship is piloted by a joyst
ick  in the first slot. If you push the redb
utton, your ship will ac";
3540 ? "celerate,";? "but becomes harder to
control, and     you may miss some asteroids!
"
3550 ? :? "  At levels of difficulty 2 and 3
, if you run into an asteroid without      c
leanly scooping it up,";
3560 ? " the resulting";? "explosion will ro
ck the galaxy, so     please be careful.";?
3570 ? "  You have only limited supplies, bu
t your ship is equipped with a timer     that
 will count down to 0."
3580 ? :? "    (hit any key to continue)";:OP
EN #Q1,Q4,Q0,"K:":GET #Q1,ANS:CLOSE #Q1
3590 GRAPHICS Q0:SETCOLOR Q2,Q4,Q10:SETCOLOR
 Q1,Q4,Q4:POKE Q752,Q1:REM POKE 16,64:POKE 5
3774,64
3600 ? "  If you are skillful enough to gath
erup all of the valuable asteroids in    one
field, you will immedi";
3610 ? "ately find";? "yourself in another,
until your timer reaches 0."
3620 ? :? "Oh, by the way, in difficulty lev
el 3,new asteroids may join the field from t
ime to time, and of";
3630 ? " course, if they";? "hit you, well,
..."
3640 ? :? "  You have been particularly sele
cted for this critical mission, and the     h
opes of all the earth ";
3650 ? "are riding";? "on your success, but
you are the best pilot on earth, and if anyo
ne can      succeed, you";
3660 ? " can! Earth desperately";? "needs th
e ores you will be mining.";? :? :? "
        GOOD LUCK!!"
3670 ? :? :? "     (hit any key to start the
game)";:OPEN #Q1,Q4,Q0,"K:":GET #Q1,ANS:CLOS
E #Q1:GOSUB 900:RETURN
3680 REM
3690 REM
```

```
3700 REM
3710 REM THESE ROUTINES POKE THE SHAPES OF T
HE PLAYERS INTO MEMORY
3720 REM
3730 REM
3740 REM
3750 RESTORE 3760:FOR A=260 TO 331:READ I:PO
KE A,I:NEXT A
3760 DATA 66,66,102,102,126,126,0,0,16,32,64
,192,193,98,60,24,63,60,48,48,48,48,60,63,24
,60,98,193,192,64,32,16
3770 DATA 0,0,126,126,102,102,66,66,24,60,70
,131,3,2,4,8,252,60,12,12,12,12,60,252,8,4,2
,3,131,70,60,24
3780 DATA 0,0,0,0,0,0,0,0
3790 FOR A=332 TO 403:READ I:POKE A,I:NEXT A
3800 DATA 0,0,0,0,0,0,24,24,0,0,0,0,0,128,19
2,224,0,0,0,192,192,0,0,0,224,192,128,0,0,0,
0,0,24,24,0,0,0,0,0,0
3810 DATA 7,3,1,0,0,0,0,0,0,0,0,3,3,0,0,0,0,
0,0,0,0,1,3,7,0,0,0,0,0,0,0,0
3820 FOR A=404 TO 475:READ I:POKE A,I:NEXT A
3830 DATA 60,24,0,0,0,0,0,0,8,12,30,15,4,0,0
,0,0,0,1,3,3,1,0,0,0,0,0,4,15,30,12,8,0,0,0,
0,0,0,24,60,0,0,0,32,240
3840 DATA 120,48,16,0,0,128,192,192,128,0,0,
16,48,120,240,32,0,0,0,0,0,0,0,0,0,0
3850 RETURN
3860 REM
3870 REM
3880 REM
3890 REM THESE ARE THE MACHINE LANGUAGE SUBR
OUTINE TO MOVE THE PLAYERS
3900 REM AND OPERATE THE COUNTDOWN TIMER
3910 REM
3920 REM
3930 REM
3940 RESTORE 3950:FOR A=Q536 TO 1560:READ I:
POKE A,I:NEXT A
3950 DATA 104,104,133,204,104,133,203,104,13
3,207,104,133,206,160,0,177
3960 DATA 206,145,203,200,192,8,208,247,96
3970 FOR I=1568 TO 1598:READ A:POKE I,A:NEXT
 I
3980 DATA 104,169,0,133,224,166,106,202,202,
202,202,202,202,202,134,225,160,255,145,224,
136,208,251,230,225,232
3990 DATA 228,106,208,242,96
```

```
4000 RETURN
4010 FOR I=1616 TO 1772:READ A:POKE I,A:NEXT
 I
4020 DATA 0,0,0,0,0,96,104,160,97,162,6,169,
6,32,92,228,96,165,88,133,208,165,89,133,209
,206,85,6,208,78,169
4030 DATA 60,141,85,6,174,82,6,202,224,255,2
40,6,142,82,6,76,188,6,162,9,142,82,6,174,81
,6,202,224,255,240,6,142
4040 DATA 81,6,76,188,6,162,5,142,81,6,174,8
0,6,169,1,141,84,6,202,224,255,240,6,142,80,
6,76,188,6,169,0,141,80,6
4050 DATA 141,81,6,141,82,6,169,1,141,83,6,1
60,179,216,162,0,240,14,169,26,145,208,200,1
89,80,6,105,16,145,208,200
4060 DATA 232,24,189,80,6,105,16,145,208,200
,232,224,3,208,228,76,95,228,104,160,95,162,
228,169,6,32,92,228,96
4070 RETURN
```

ASTEROID MINERS

A UNIQUE GAME TUTORIAL

by Mark Chasin

Source Code for the

Machine Language

Subroutines

```
                0100 ;
                0110 ;
                0120 ;
                0130 ;***************************
                0140 ;***************************
                0150 ;***  ASTEROID MINERS   ***
                0160 ;***   CHARACTER SET    ***
                0170 ;*** RELOCATING ROUTINE***
                0180 ;***  The Source Code   ***
                0190 ;***                    ***
                0200 ;*** (c)1982 MMG MICRO ***
                0210 ;***      SOFTWARE      ***
                0220 ;***                    ***
                0230 ;***   by Mark Chasin   ***
                0240 ;***                    ***
                0250 ;***************************
                0260 ;***************************
                0270 ;
                0280 ;
                0290 ;
0000            0300          *=   $0600
                0310 ;
0600 68         0320          PLA            ;FOR BASIC
0601 68         0330          PLA            ;WHERE TO
0602 85CC       0340          STA  $CC       ;PUT THE
0604 68         0350          PLA            ;CHARACTER
0605 85CB       0360          STA  $CB       ;SET
                0370 ;
0607 A900       0380          LDA  #0
0609 85CD       0390          STA  $CD       ;COMES
060B A9E0       0400          LDA  #$E0      ;FROM
060D 85CE       0410          STA  $CE       ;$E000
060F A204       0420          LDX  #4        ;4 PAGES
                0430 ;
0611 A000       0440 A2       LDY  #0
0613 B1CD       0450 A1       LDA  ($CD),Y   ;MOVE
0615 91CB       0460          STA  ($CB),Y
0617 C8         0470          INY
0618 D0F9       0480          BNE  A1        ;DONE?
061A E6CC       0490          INC  $CC       ;NEXT
061C E6CE       0500          INC  $CE       ;PAGE
061E CA         0510          DEX
061F D0F0       0520          BNE  A2        ;DONE?
0621 60         0530          RTS            ;RETURN
```

```
               ;
               ;
               ;
               ;******************************
               ;******************************
               ;*** ASTEROID MINERS ROUTINE ***
               ;***   TO MOVE THE PLAYERS   ***
               ;***                         ***
               ;***     The Source Code     ***
               ;***                         ***
               ;***    (c)1982 MMG MICRO    ***
               ;***        SOFTWARE         ***
               ;***                         ***
               ;***      by Mark Chasin     ***
               ;***                         ***
               ;******************************
               ;******************************
               ;
               ;
               ;
0000                    *=   $0600
0600 68                 PLA                 ;NEEDED BY BASIC
0601 68                 PLA                 ;GET WHERE TO PUT THE
0602 85CC               STA  $CC            ;PLAYER-STORE INTO
0604 68                 PLA                 ;$CB,$CC
0605 85CB               STA  $CB
0607 68                 PLA                 ;NOW STORE WHICH
0608 85CF               STA  $CF            ;SHAPE INTO $CE,$CF
060A 68                 PLA
060B 85CE               STA  $CE
060D A000               LDY  #0
060F B1CE       NXTBYT  LDA  ($CE),Y      ;GET SHAPE BYTE
0611 91CB               STA  ($CB),Y        ;PUT IN POSITION
0613 C8                 INY
0614 C008               CPY  #8             ;GET 8 BYTES
0616 D0F7               BNE  NXTBYT         ;DONE? NO, BRANCH
0618 60                 RTS                 ;YES. RETURN
```

```
                0110 ;
                0120 ;
                0130 ;******************************
                0140 ;******************************
                0150 ;*** ASTEROID MINERS ROUTINE ***
                0160 ;***  TO CLEAR THE TOP EIGHT ***
                0170 ;***PAGES OF MEMORY TO ZEROS ***
                0180 ;***                        ***
                0190 ;***     The Source Code    ***
                0200 ;***                        ***
                0210 ;***    (c)1982 MMG MICRO   ***
                0220 ;***         SOFTWARE       ***
                0230 ;***                        ***
                0240 ;***      by Mark Chasin    ***
                0250 ;***                        ***
                0260 ;******************************
                0270 ;******************************
                0280 ;
                0290 ;
                0300 ;
0000            0310        *=   $0620
0620 68         0320        PLA                 ;NEEDED BY BASIC
0621 A900       0330        LDA  #00            ;SET UP $CA AND
0623 85CA       0340        STA  $CA            ;$CB AS AN INDIRECT
0625 A66A       0350        LDX  $6A            ;ADDRESS TO THE TOP
0627 CA         0360        DEX                 ;OF MEMORY MINUS
0628 CA         0370        DEX                 ;SEVEN
0629 CA         0380        DEX
062A CA         0390        DEX
062B CA         0400        DEX
062C CA         0410        DEX
062D CA         0420        DEX
062E 86CB       0430        STX  $CB
0630 A0FF       0440 NWPAGE LDY  #$FF        ;TO CLEAR THE WHOLE PAGE
0632 91CA       0450 AGAIN  STA  ($CA),Y      ;PUT ZERO THERE
0634 88         0460        DEY                 ;IF WE FINISH THE PAGE,
0635 D0FB       0470        BNE  AGAIN          ;DON'T BRANCH, BUT
0637 E6CB       0480        INC  $CB            ;DO NEXT PAGE
0639 E8         0490        INX
063A E46A       0500        CPX  $6A            ;UNLESS WE'VE FINISHED
063C D0F2       0510        BNE  NWPAGE
063E 60         0520        RTS                 ;THEN RETURN
```

```
              ;
              ;
              ;
              ;****************************
              ;****************************
              ;***  ASTEROID MINERS TIMER ***
              ;***     The Source Code     ***
              ;***                         ***
              ;***    (c)1982 MMG MICRO    ***
              ;***        SOFTWARE         ***
              ;***                         ***
              ;***     by Mark Chasin      ***
              ;***                         ***
              ;****************************
              ;****************************
              ;
              ;
              ;
              ;THE EQUATES USED IN THIS FILE
              ;
              ;
              ;
0058          SAVMSC =    $58
0222          VVBLKI =    $222
E45F          SYSVBV =    $E45F
00D0          SCREEN =    $D0
001A          COLON  =    26
00B3          OFFSET =    179
0010          CONVER =    16
E45C          SETVBV =    $E45C
              ;
              ;
              ;
              ;THE STORAGE LOCATIONS FOR THE TIMER
              ;DIGITS AND INDICATORS
              ;
              ;
              ;
0000                 *=   $0650
0650 00       TIMDIG .BYTE 0,0,0
0651 00
0652 00
0653 00       INDIC  .BYTE 0
0654 00       INDIC1 .BYTE 0
0655 3C       TIMER  .BYTE 60
              ;
              ;
              ;
              ;THE ROUTINE TO INSERT THE TIMER ROUTINE
              ;INTO THE VERTICAL BLANK INTERRUPT
              ;PROCESSING, CREATING AN EFFECTIVE
```

```
               0590 ;MULTIPROCESSOR FROM YOUR ATARI!
               0600 ;
               0610 ;
               0620 ;
0656 68        0630        PLA
0657 A061      0640        LDY   #CLOCK&255
0659 A206      0650        LDX   #CLOCK/256
065B A906      0660        LDA   #6
065D 205CE4    0670        JSR   SETVBV
0660 60        0680        RTS
               0690 ;
               0700 ;
               0710 ;
               0720 ;THE ACTUAL TIMER ROUTINE ITSELF
               0730 ;
               0740 ;
               0750 ;
0661 A558      0760 CLOCK  LDA   SAVMSC
0663 85D0      0770        STA   SCREEN
0665 A559      0780        LDA   SAVMSC+1
0667 85D1      0790        STA   SCREEN+1
0669 CE5506    0800        DEC   TIMER
066C D04E      0810        BNE   PRINT
066E A93C      0820        LDA   #60
0670 8D5506    0830        STA   TIMER
0673 AE5206    0840 DIGSEC LDX   TIMDIG+2
0676 CA        0850        DEX
0677 E0FF      0860        CPX   #$FF
0679 F006      0870        BEQ   STORE2
067B 8E5206    0880        STX   TIMDIG+2
067E 4CBC06    0890        JMP   PRINT
0681 A209      0900 STORE2 LDX   #9
0683 8E5206    0910        STX   TIMDIG+2
0686 AE5106    0920        LDX   TIMDIG+1
0689 CA        0930        DEX
068A E0FF      0940        CPX   #$FF
068C F006      0950        BEQ   STORE1
068E 8E5106    0960        STX   TIMDIG+1
0691 4CBC06    0970        JMP   PRINT
0694 A205      0980 STORE1 LDX   #5
0696 8E5106    0990        STX   TIMDIG+1
0699 AE5006    1000        LDX   TIMDIG
069C A901      1010        LDA   #1
069E 8D5406    1020        STA   INDIC1
06A1 CA        1030        DEX
06A2 E0FF      1040        CPX   #$FF
06A4 F006      1050        BEQ   STOREM
06A6 8E5006    1060        STX   TIMDIG
06A9 4CBC06    1070        JMP   PRINT
06AC A900      1080 STOREM LDA   #0
06AE 8D5006    1090        STA   TIMDIG
06B1 8D5106    1100        STA   TIMDIG+1
```

```
06B4 8D5206 1110        STA  TIMDIG+2
06B7 A901   1120        LDA  #1
06B9 8D5306 1130        STA  INDIC
06BC A0B3   1140 PRINT  LDY  #OFFSET
06BE D8     1150        CLD
06BF A200   1160        LDX  #0
06C1 F00E   1170        BEQ  DIGIT2
06C3 A91A   1180 PRNTLP LDA  #COLON
06C5 91D0   1190        STA  (SCREEN),Y
06C7 C8     1200        INY
06C8 BD5006 1210        LDA  TIMDIG,X
06CB 6910   1220        ADC  #CONVER
06CD 91D0   1230        STA  (SCREEN),Y
06CF C8     1240        INY
06D0 E8     1250        INX
06D1 18     1260 DIGIT2 CLC
06D2 BD5006 1270        LDA  TIMDIG,X
06D5 6910   1280        ADC  #CONVER
06D7 91D0   1290        STA  (SCREEN),Y
06D9 C8     1300        INY
06DA E8     1310        INX
06DB E003   1320        CPX  #3
06DD D0E4   1330        BNE  PRNTLP
06DF 4C5FE4 1340 EXIT   JMP  SYSVBV
            1350 ;
            1360 ;
            1370 ;
            1380 ;ROUTINE TO TURN OFF THE CLOCK BY RESETTING
            1390 ;THE VERTICAL BLANK INTERRUPT VECTOR TO ITS
            1400 ;ORIGINAL VALUE
            1410 ;
            1420 ;
            1430 ;
06E2 68     1440        PLA
06E3 A05F   1450        LDY  #$5F
06E5 A2E4   1460        LDX  #$E4
06E7 A906   1470        LDA  #6
06E9 205CE4 1480        JSR  SETVBV
06EC 60     1490        RTS
```